IFT6285 (Devoir 3) - Université de Montréal

COLELLA Charles

October 9, 2021

1 Introduction

Dans ce devoir nous allons aborder la correction de mots comme un problème d'identification des voisins d'un mot erroné, le candidat à la correction étant sélectionné dans ce voisinage. Pour mesurer la différences entre deux séquences, nous utiliserons diverses métriques que nous comparerons dans ce document.

1.1 Distance de Levenshtein

La distance de **Levenshtein** entre deux mots est le nombre minimum de modifications à caractère unique (insertions, suppressions ou substitutions) nécessaires pour changer un mot en l'autre.
Etant donné $a, b \in M$ avec M l'ensemble des mots. En designant par $a-1$ le mot a tronqué de sa premiere lettre, la distance de **Levenshtein** est décrite de maniere récursive ci-dessous par :

$$\text{lev}(a,b) = \begin{cases} \max(|a|,|b|) & \text{si } \min(|a|,|b|) = 0, \\ \text{lev}(a-1,b-1) & \text{si } a[0] = b[0], \\ 1 + \min \begin{cases} \text{lev}(a-1,b) \\ \text{lev}(a,b-1) \\ \text{lev}(a-1,b-1) \end{cases} & \text{sinon.} \end{cases} \tag{1}$$

Il existe aussi la distance de **Damerau–Levenshtein** qui reprend le principe de celle de Levenshtein mais en y ajoutant les transpositions.

1.2 Distance de Jaro

La distance de **Jaro** prend en compte l'éloignement des caractères indentiques dans les deux chaines ainsi que le nombre de transpositions nécéssaires.

$$d_j(a,b) = \frac{1}{3}\left(\frac{m}{|a|} + \frac{m}{|b|} + \frac{m-t}{m}\right) \tag{2}$$

Où m représente le nombre de correspondances (deux caractères sont considérés comme correspondants si leur éloignement est inférieur à la moitié de la taille du plus grand mot).
Le nombre de fois où le i-*ème* caractère de a est différent du i-*ème* caractère de b, divisé par deux, donne le nombre de transpositions t.

La distance **Jaro-Winkler** utilise une échelle de préfixe $p \leq 0.25$ ce qui donne des notes plus favorables aux chaînes qui correspondent dès le début pour une longueur de préfixe définie $l \leq 4$:

$$d_{jw}(a,b) = d_j(a,b) + \ell p(1 - d_j(a,b)) \tag{3}$$

2 Application à la correction orthographique

2.1 Mise en place

Pour mettre en place un système de correction orthographique avec n propositions de corrections, il suffit d'évaluer la distance du mot à corriger avec ceux d'un dictionnaire et à en prelever les n mots dont la distance est la plus faible. Le dictionnaire utilisé **ici**[1] correspond aux 201 315 mots rencontrés au moins 16 fois dans la partie d'entrainement du corpus **1Bwc**[2] avec, pour chacun, leur fréquence d'apparition. Ainsi si deux mots ont une distance identique, celui qui sera selectionné en premier sera celui avec la fréquence d'apparition la plus élevée.

Pour évaluer la veracité des corrections proposées, nous dispons d'un **fichier**[3] contenant 3935 mots erronés ainsi que leur correction. Un système de score a donc été mis en place afin de mesurer l'efficacité des correcteurs: Soit c_k la correction préalable du mot numéro k, $t_{k,j}$ la correction proposée à la position j pour le k-*ème* mots de la liste et s_k le score total au moment de l'évaluation du k-*ème* mots de la liste.

$$s_{k+1} = \begin{cases} s_k - \frac{dlev(c_k, t_{k,0})}{10} & \text{si aucun des mots proposé n'est le bon} \\ s_k + \frac{1}{1+j} & \text{si le mot proposé à la position } j \in [\![0, n-1]\!] \text{ est le bon} \end{cases} \tag{4}$$

dlev correspond à la distance de **Damerau–Levenshtein** qui prend en compte le nombre de transformations à réaliser pour passer d'un mot à l'autre.

2.2 Résultats

Différentes distances ont été testées :

- Distance de **Hamming** (qui mesure le nombre de caractères différents entre deux mots)
- Distance de **Levenshtein** et de **Damerau–Levenshtein**
- Distance de **Jaro** et de **Jaro-Winkler** ($l = 3$, $p = 0.1$)

Métrique	Damerau–Levenshtein	Levenshtein	Jaro	Jaro-Winkler	Hamming
Corrections justes	3332	3215	2915	2928	1159
Corrections erronées	603	720	1020	1007	2776
Score total	2480.5	2288	1774	1795	−977
Temps total (s)	2311	988	10180	13001	2048
Efficacité (%)	84.7	81.7	74.1	74.4	29.5

Dans cette expérience, trois corrections ont été proposées pour chaque distance étudiée. Si une des trois corrections est correcte, elle est comptabilisée comme juste.
En décomposant le score total en la somme des bonus et la somme des malus on obtient le tableau ci-dessous pour les deux meilleurs scores :

Métrique	Damerau–Levenshtein	Levenshtein
Total bonus	2894.5	2757
Total malus	414	469
Moyenne valeur bonus	0.87	0.86
Moyenne valeur malus	0.69	0.65

Bien que générant une très légere différence dans les moyennes, les résultats sont relativement proches. Les deux algorithmes trouvent en moyenne plutôt lors du premier choix lorsqu'ils réussissent. D'autre part lorsque les traductions proposées ne sont pas correctes, la solution est en moyenne entre 6 et 7 transformations de la premiere correction proposée (petit avantage pour **Levenshtein**).

[1] http://www-labs.iro.umontreal.ca/~felipe/IFT6285-Automne2020/voc-1bwc.txt
[2] https://github.com/ciprian-chelba/1-billion-word-language-modeling-benchmark
[3] http://www-labs.iro.umontreal.ca/~felipe/IFT6285-Automne2020/devoir3-train.txt

On peut afficher l'évolution du score pour chaque distance :

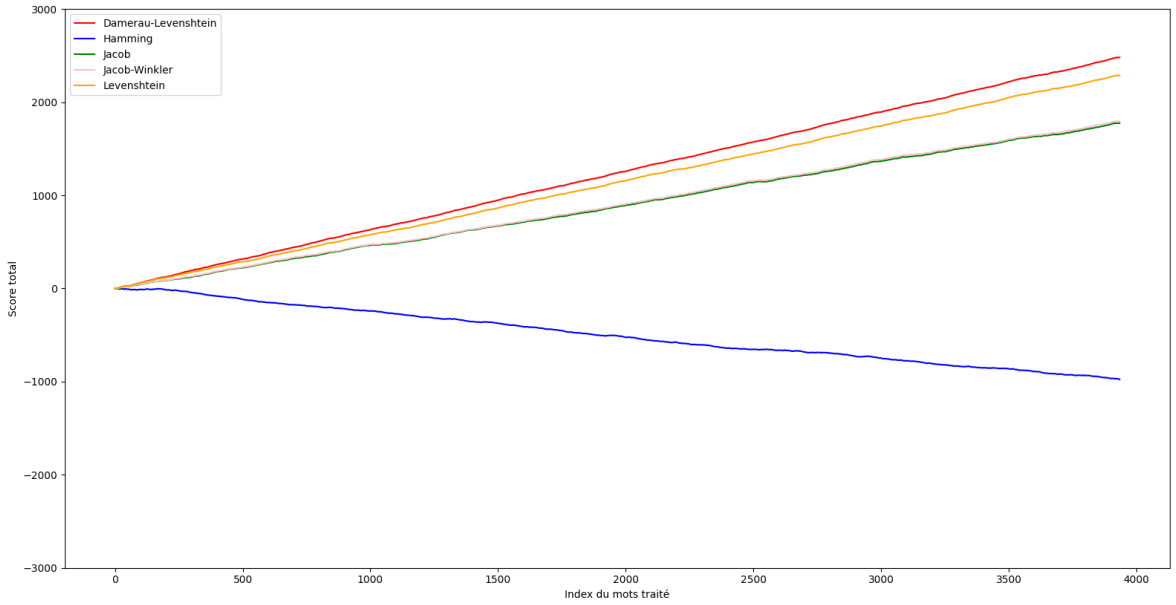

Figure 1: Score en fonction du nombre de mots traités

Les différentes distances ont été implémentées avec les librairies **python-Levenshtein**[4], **fastDamerauLevenshtein**[5], **scipy**[6]et **pyjarowinkler**[7]. L'ordinateur de test comporte un Intel Core i5-10300H 2.50GHz avec 16Go de RAM.

3 Choix du meilleur modèle

3.1 Mise en place

Les résultats précédents mettent en avant que, malgré une efficacité relativement proche, l'algorithme de **Levenshtein** dure 58.3% moins de temps que celui de **Damerau–Levenshtein**.
Une premiere idée serait de réduire le temps de calcul de l'algorithme de Damerau–Levenshtein en ne l'appliquant que sur α mots choisis au préalable avec la distance de **Levenshtein**. On obtient le tableau ci-dessous en faisant varier α :

α	10	20
Score total	2423	2463.5
Moyenne valeur bonus	0.87	0.87
Moyenne valeur malus	0.66	0.67
Temps total (s)	1425	1737
Efficacité (%)	83.3	84.2

Les performances pour $\alpha = 20$ sont très proches de celles de l'algorithme de **Damerau–Levenshtein** sans séléction au préalable (84.7 % d'efficacité contre 84.2% ici) et cette nouvelle solution engendre un gain de temps de 25% par rapport à l'algorithme précédent. Pour $\alpha = 10$, le temps de calcul diminue de 39% pour un écart de score de seulement 3% (contre 8% pour Levenshtein).

[4] https://pypi.org/project/python-Levenshtein/
[5] https://pypi.org/project/fastDamerauLevenshtein/
[6] https://www.scipy.org/
[7] https://pypi.org/project/pyjarowinkler/

Etant donné que la distance de **Jaro-Winkler** est particulièrement adaptée au traitement de chaînes courtes, une seconde idée serait d'utiliser la distance de **Jaro-Winkler** pour les chaines de taille inférieure à $\lambda \in N^*$ et celle de **Damerau–Levenshtein** pour les autres (en gardant la séléction préalable avec **Levenshtein**). Pour $\alpha = 10$, $\lambda = 6$, $p = 0.1$, et $l = 4$ le score obtenu est de seulement 1918 (en 1400s), nous n'approfondiront donc pas cette méthode.

4 Limites de l'approche utilisée

Au cours de ce document, différentes distances ont été étudiées puis comparées. Ces méthodes évaluent uniquement un nombre de transformations applicables pour faire concorder un mot avec un autre du dictionnaire. Bien que la conjugaison reste simple en anglais (langue utilisée pour les différents tests de ce devoir), appliquer les mêmes distances pour corriger un mot en français deviendrait beaucoup plus compliqué. Il faudrait que le dictionnaire contiennent toutes les conjugaisons d'un mot donné : *mange-mangais-mangerai*-ect... Cette contrainte imposerait une taille très conséquente du dictionnaire.

De plus, déterminer la correction d'un mot releve aussi du contexte d'une phrase, pas seulement de l'orthographe. Par exemple *mangeair* est à une transformation de *mangeait*, une de *mangeais*, donc laquelle choisir ? En se rapportant aux conjugaisons, une seule faute de frappe pourrait par ailleurs rendre la correction d'un mot complétement erronée (cf : moyenne valeur malus pour Damerau-Levenshtein et Levenshtein qui montre une erreur située entre 6 et 7 transformations lorsque les corrections ne sont pas justes).

Enfin on remarque que l'approche utilisée est très coûteuse en performances (cf : temps total pour chaque distance), parcourir tout un dictionnaire pour chaque mot est très long (d'autant plus que nous disposions de seulement 201315 mots anglais dans notre dictionnaire). En élargissant ce processus à des tâches plus longues, comme la lecture d'un corpus entier, cette technique serait très ardue étant donné qu'il faudrait au moins parcourir le dictionnaire sur chaque mot pour évaluer si une faute est présente.